Test Coverage

In software testing, test coverage is a metric which measures the amount of testing done by a set of tests and how much of the program is actually tested once they are ran. The bigger the coverage the better, as it increases the chances of discovering faulty code leading us to fix the errors and improve the overall quality of the application.

As 100% test coverage is impossible due to various constraints (time being the biggest factor in our case) developers need to select the most important scenarios to be covered as much as possible. The most important scenarios (high risk ones) are the ones which will be accessed/used the most.

3.2.2.     Risk Analysis

In order to ensure that our application does not encounter any major defects and solution runs as smoothly as possible, we have conducted numerous peer reviews to determine which scenarios are most likely to occur. We used a risk matrix to categorize and visualize the risk/priority a given scenario happens to be in. A risk level is chosen based on likelihood of it occurring the impact it has on the software. Each risk level is color coded for less ambiguity

|  |  |  |  |
| --- | --- | --- | --- |
| Likelihood -> | Impact -> | | |
| Low | Medium | High |
| Low | Medium | Medium |
| Low | Low | Low |

Scenarios are ranked as follows:

* The scenarios with the highest risk rating are given full priority as they are vital to the workings of the application and their test coverage must be as close to fully covered as possible (over 80%)
* Medium risk rating is considered to be important but not a vital scenario, so the test coverage while must be adequate, does not require to be fully covered
* Scenarios with low risk ratings are to be covered only to the extent that ensures the user does not run into problems as long as they do not deviate from the preplanned usage path.

Below are some examples of various scenarios with their corresponding risk levels

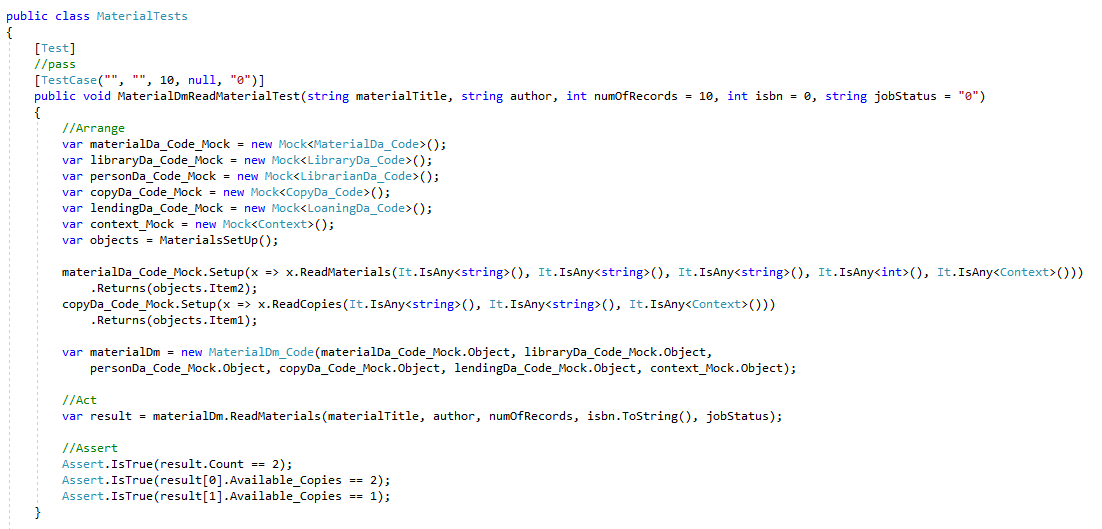
|  |  |
| --- | --- |
| Scenario | Risk level |
| Checking out a book when not allowed | High |
| Returning a book which was not loaned | High |
| Adding new books to the library | Medium |
| Removing old books from the library | Medium |
| Renewal of membership card even if the person is not a member anymore | Low |
| Removal of member who is not a member anymore | Low |

Test implementation

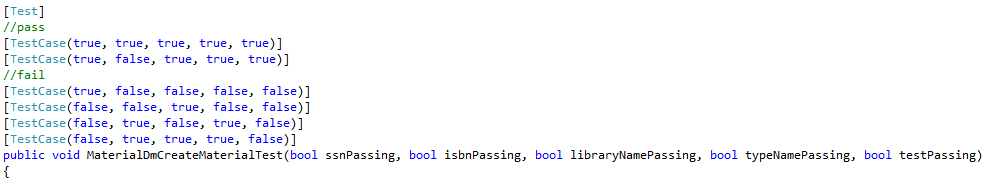
In this section we will talk about how we have implemented tests in our project and why each level of testing is integral to a high-quality final product.

Unit tests

Unit tests are the smallest “unit” of tests. These tests are made to validate if a single individual unit of code functions as expected. These tests are basic and have no logic, they consist of a single or a few inputs and produce a single output. These tests have the greatest effect on quality.



In the image above (figure x) we have conducted a test on materials. In this test we return a book and see if the results comply with our standards. We also mocking the methods which are called by the test method.

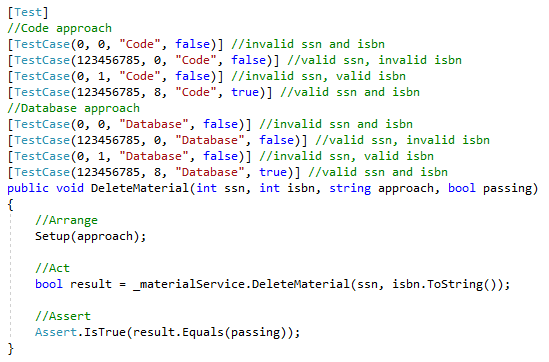


If the result complies with a favorable test case, it passes. If any parameter of the result is returned as false, it does not pass.

These tests are the first level of testing developers carry out before moving on to integration testing

Integration tests

After the units of tests have been verified to be working in correct order, they are ready to be integrated into the application. At this level the units are combined, and they are tested as a group, where we search for defects when they are interacting with the already integrated units of code.



In this test we test the code by giving invalid attributes and see if we are thrown an error or not. We expect this test to pass and if the value in the passing parameter to be true, it means the unit of code has been successfully integrated and functions as intended.

System testing

After the successful integration tests, system tests are meant to be run when the integrated software is ready to be tested as a whole and measured against the initial requirements.

Acceptance tests

The final level of software testing which is meant to test the software’s compliance to the original business requirements. These tests in contrast to the previous ones, are best performed by a select group of individuals who represent the final users of the product. If the software passes acceptance tests it is ready to be launched and go commercial.



In this test we mock a user logging into the system. If the result of the test is the home page, the test has been successful.

When to stop testing?

There are no specific rules as to when to stop testing, the only rational one being when everything has been covered, but as mentioned that option is not feasible in a project of this size.

The next best thing is to look at the feasibility of the option of finding the next defect, does the cost of finding the next error exceed the expected loss, if it does and the development team is satisfied with the quality and coverage goals have been met, it is a viable option to stop testing and release the application

Another reason to stop is when the project leader or the boss says the necessary requirements have been met and the application is ready to be shipped.